Pekka Kulju

OHJELMISTOTUOTANNON PROSESSI PELIALAN YRITYKSESSÄ

OHJELMISTOTUOTANNON PROSESSI PELIALAN YRITYKSESSÄ

Pekka Kulju
Opinnäytetyö
Kevät 2015
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehitys

Tekijä(t): Pekka Kulju
Opinnäytetyön nimi: Ohjelmistotuotannon prosessi pelialan yrityksessä
Työn ohjaaja(t): Lasse Haverinen
Työn valmistumislukukausi ja -vuosi: Kevät 2015 Sivumäärä: *37 + 2 liitettä*

Opinnäytetyö tehtiin aloittavan pelialan yrityksen tuotannonprosessin kartoittamiseksi. Työn tavoitteena oli selvittää ohjelmistotuotannonprosessi pelialla ja soveltaa se yrityksen tarpeisiin. Yrityksellä ei ollut opinnäytetyötä kirjoitettaessa paljoa resursseja henkilöstöpuolella, mikä vaikutti työn lopputulokseen.

Työssä otettiin huomioon tavanomaiset ohjelmistotuotannon prosessimallit ja pelialalle jo sovelletut mallit. Tämän lisäksi työssä pohdittiin prosessin ylläpitämiseen käytettäviä työkaluja.

Työn tuloksena huomattiin eroavaisuus tavanomaiseen ohjelmistotuotantoon ja muutettiin se yritykselle sopivaksi. Työn tulokset olivat hyvin läheltä pelialalla jo käytettyjä menetelmiä, mutta tavanomaiset tuotantoprosessit eivät käyneet sellaisenaan yrityksen tarpeisiin.

Asiasanat: tuotantoprosessi, ohjelmistosuunnittelu, peliteollisuus

SISÄLLYS

1 JOHDANTO

Olin opinnäytetyötä kirjoittaessani perustamassa pelialan yritystä. Yrityksellä ei ollut paljoa resursseja, mikä johti useisiin ongelmiin tuotannon järjestelyissä. Ennen opinnäytetyötä olin yrittänyt useamman kerran kehittää yritykselleni julkaisukelpoista peliä huonolla menestyksellä. Viimeisen yrityksen jälkeen ongelma saatiin keskitettyä projektin hallinnan kasvavaan hankaluuteen projektin edetessä. Tämän lisäksi pelialan tuotantoprosessi eroaa tavanomaisesta ohjelmistotuotantoprosessista jonkin verran.

Tämän opinnäytetyön tavoitteena oli selvittää, minkälainen tuotannon järjestely sopisi parhaiten aloittavan pelialan yrityksen tarpeisiin. Opinnäytetyössä keskitytään pääsääntöisesti työn suunnitteluun, tuotantoon ja sen ylläpitoon käsillä olevilla resursseilla. Näissä vaiheissa selvitetään, miten työn voisi toteuttaa mahdollisimman selkeästi ja miten ne sitoutuvat toisiinsa. Opinnäytetyö ei sisällä ohjelmointia, mutta ohjelmoinnin ulkoasuun puututaan.

Opinnäytetyössä ei puututa ohjelmoinnin ulkopuolella tapahtuvaan tuotantoon, kuten grafiikkaan. Myöskään tuotteen markkinointiin ei tarkemmin puututa. Tästä huolimatta niistä voidaan mainita erikseen, jos sen koetaan selkeyttävän senhetkistä aihetta.

2 OHJELMISTOTUOTANTOPROSESSI

Ohjelmistotuotannolla tarkoitetaan ohjelmistokehityksen hallintaa ja suunnittelua. Ideana on luoda pohja ohjelmistotyölle, jonka tuloksena syntynyt järjestelmä täyttää käyttäjien kohtuulliset toiveet ja odotukset aikataulujen ja kustannusarvioiden puitteissa. (1.) Ohjelmistotuotantomenetelmiä kutsutaan vaihejakomalleiksi.

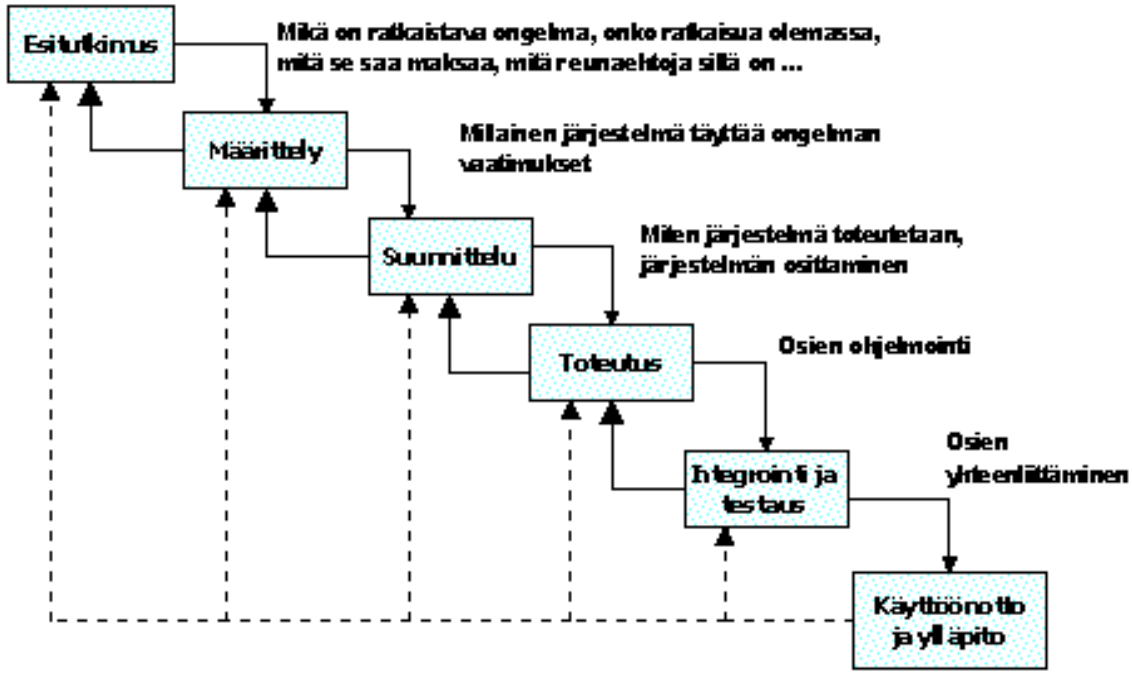
2.1 Vaihejakomallit

Ohjelmistotuotannolle on kehitetty useita erilaisia vaihejakomalleja. Vaihejakomallien avulla kehitystyön koko elinkaari jaetaan vaiheisiin. Erilaisia vaihejakomalleja ovat muun muassa vesiputousmalli, joka on lineaarinen, EVO-malli, joka on iteratiivinen eli toistuva ja inkrementaalinen eli lisäävä, protoilumalli, jossa tuotetaan eräänlainen esituote ennen varsinaisen tuotteen luomista, ja Agile-malli eli ketterä menetelmä, joka sisältää useamman iteratiivisen mallin. Jokainen malli on sovellettava yrityksen tai projektin omiin tarpeisiin. (1.) Opinnäytetyössä keskitytään vertailemaan tavanomaista vesiputousmallia ja uusinta ketterää menetelmää Scrumia, joista voidaan sanoa, että kumpikin edustaa ohjelmistotuotannon ääripäitä. Muut mallit osuvat kutakuinkin näiden mallien väliin, ja niihin voidaan tarpeen tullen viitata.

2.1.1 Vesiputousmalli

Tavanomaisista vaihejakomalleista ehkä käytetyin on vesiputousmalli. Vesiputousmallissa luodaan kattava dokumentaatio ennen ohjelmoinnin aloittamista. Vesiputousmallin dokumentaatiolla pyritään ratkaisemaan ohjelman mahdolliset rakenteelliset ongelmat ennen tuotantoa, ja tuotantoon päästyä pyritään ohjelma saamaan suoraan testaukseen ja julkaisuun. Vesiputousmallissa vaiheita ei toteuteta päällekkäin vaan yksi kerrallaan. Tästä johtuen, kun päästään tuotantoon, on hyvin vaikea palata enää suunnittelupöydän ääreen. (1.)

Vesiputousmallin etuna on se, että se on yksinkertainen ja helppo toteuttaa. Tämä sopii erityisen hyvin pieniin projekteihin, joissa projektin vaatimukset tiedetään hyvin. Vesiputousmallin vaiheet ovat esitutkimus, määrittely, suunnittelu, tuotanto ja testaus. Vesiputousmallin eteneminen on havainnollistettu kuvassa 1.



KUVA 1. Vesiputousmallin vaihejakomalli (2)

Vesiputousmallin **esitutkimuksessa** laaditaan lyhyt alustava selvitys potentiaalisesta projektista. Tarkoituksena on vastata kysymykseen, miksi tai miksi ei järjestelmä(ä) tulisi tehdä. Esitutkimusvaiheessa laaditaan järjestelmälle vaatimukset laitteiston ja asiakkaan vaatimusten mukaan. Esitutkimuksen jälkeen tehdään päätös, aloitetaanko projekti, lykätäänkö se vai hylätäänkö se kokonaan. (1.)

Määrittelyvaiheessa tehtävänä on ottaa selville, mitä ollaan tekemässä. Tässä vaiheessa laaditaan esitutkimusvaiheessa saaduista vaatimuksista kuva järjestelmästä asiakkaan näkökulmasta katseltuna. Määrittelyvaiheessa ei ole tarpeen kuormittaa lukijaa liiallisilla yksityiskohdilla. Määrittelytyön avainasemassa ovat oikein ymmärretyt ja mahdollisimman muuttumattomina pysyneet asiakasvaatimukset. (1.)

Suunnitteluvaiheessa pohditaan määrittelyvaiheessa esille tulleiden ominaisuuksien käytännön toteutusta. Tarkoituksena on muuttaa määrittelyvaiheessa saadut ominaisuudet tekniseksi ja vastata kysymykseen, miten projekti tullaan toteuttamaan. Järjestelmä pilkotaan niin pieniin osiin, että saadut komponentit voidaan antaa yksittäisten suunnittelijoiden tehtäviksi. Suunnitteluvaiheen tuloksena on dokumentaatio järjestelmän arkkitehtuurista. Siinä kuvataan järjestelmän fyysinen arkkitehtuuri ja sen käyttämät komponentit. Kuvauksen kautta siirrytään yksittäisiin komponentteihin ja niiden rakenteisiin ja toteutuksiin. Dokumentoidaan siis ratkaisun filosofia eli kaikki moduulit ja niiden väliset yhteydet ja rajapinnat. (1.)

Toteutusvaiheessa aletaan luomaan järjestelmää aikaansaatujen dokumenttien pohjalta. Vesiputousmallissa toteutus pyritään saamaan kerralla valmiiksi ja integroinnissa ja testauksessa kootaan saadut toteutukset valmiiksi järjestelmäksi.

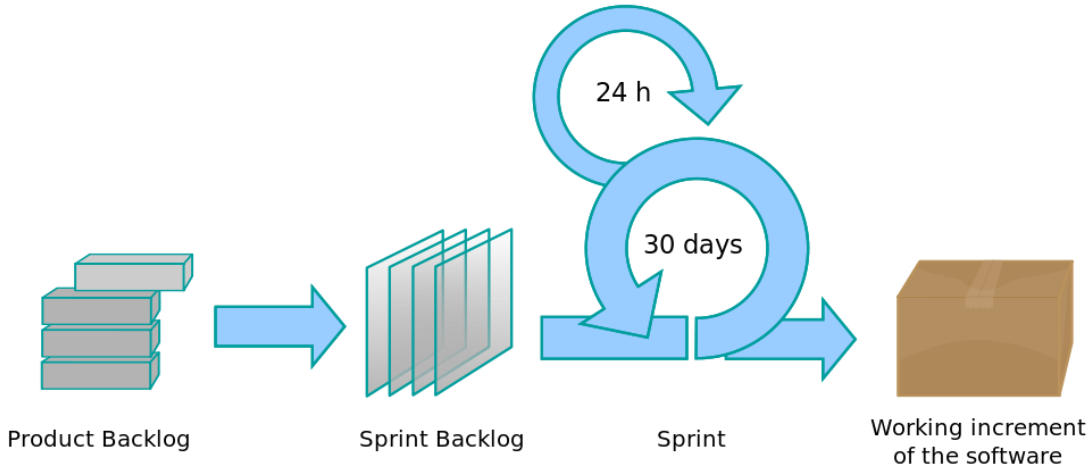
2.1.2 Ketterät menetelmät

Ketterät menetelmät perustuvat kevyempään lähestymistapaan lyhyillä kehitysykleillä ja pyrkivät antamaan kehitystyölle jatkuvaa palautetta. Ketterien menetelmien vahvuudet perustuvat tuotteen toimivien versioiden nopeaan julkaisuun, käyttäjäläheiseen menettelytapaan ja nopeaan reagointikykyyn muuttuvissa olosuhteissa (3).

Ketterillä menetelmillä on puolestaan omat heikkoutensa. Muun muassa suurissa projekteissa on vaikeaa määritellä projektin vaativuutta, ja ketterissä me-

netelmissä puuttuu painotus vaaditulle suunnittelulle ja dokumentoinnille. Projekti voi harhautua tavoitteistaan, jos asiakasedustaja ei ole täysin selvillä, millaista lopputulosta he haluavat. Vain kokeneet ohjelmoijat kykenevät tekemään järkeviä päätöksiä kehitystyössä. (3.) Näin ollen ketterät menetelmät eivät välttämättä sovi projekteille, jossa vaaditaan täsmällistä toteutusta, esimerkiksi jos tuotteen varaan lasketaan ihmishenkiä.

Ketteristä menetelmistä kuuluisin ja käytetyin on Scrum. Toisin kuin tavanomainen ohjelmistotuotantomalli, Scrum ei sisällä kattavaa dokumentaatiota siitä, miten projekti tulisi toteuttaa, vaan projektiryhmälle annetaan kuvaus halutusta lopputuloksesta ja toteutus jätetään laajalti kehittäjien vastuulle. Scrum perustuu iteratiiviseen ja lisäykselliseen malliin, jossa tuotanto toteutetaan enintään kuukauden pituisissa kehityssykleissä, joita kutsutaan sprinteiksi. (4.) Scrumin eteneminen on kuvattu kuvassa 2.



KUVA 2 Scrumin eteneminen (5)

Scrumin sisältämät tuotokset pyrkivät tarjoamaan projektille läpinäkyvyyttä siten, että koko projektiryhmällä on sama ymmärrys projektin rakenteesta. Nämä tuotokset ovat Product backlog eli tuotteen kehitysjono, Sprint backlog eli sprintin tehtävälista ja increment eli tuoteversio. (4.)

Tuotteen kehitysjono sisältää eräänlaisen tilauslistan kaikesta, jota saatetaan tarvita tuotteessa, ja se toimii ainoana lähteenä kaikille tuotteen muutoksille. Tuotteen kehitysjono sisältää kaikki tuotteen ominaisuudet, toiminnot, vaatimukset, parannukset ja korjaukset, jotka ovat mukana tuotteeseen tulevissa julkaisuissa. Tuotteen kehitysjono ei ole koskaan valmis, vaan se toimii dynaamisesti muuttuen jatkuvasti vastaamaan tuotteen tarpeita. Kehityksen ensiaskelilla tuotteen kehitysjono sisältää vain tiedetyt ja parhaiten ymmärretyt vaatimukset. (4.)

Sprintin tehtävälista sisältää valikoiman asiakohtia tuotteen kehitysjonosta, jotka sisällytetään seuraavaan sprinttiin. Sen lisäksi se sisältää suunnitelman tuotteen tuoteversion kehitykselle ja sprintin tavoitteen määrittelyn. Sprintin tehtävälista on tarpeeksi tarkka suunnitelma, että se ymmärretään päivittäisissä palavereissa. Sprintin tehtävälista elää koko sprintin läpi. Kehitysryhmä lisää sprintin tehtävälistaan kaikki asiakohdat, jotka tarvitaan sprintin tavoitteiden saavuttamiseksi, ja tarvittaessa poistaa turhat asiakohdat. Vain kehittäjät voivat lisätä tai poistaa kohtia sprintin tehtävälistasta. (4.)

Tuoteversio on kaikkien tuotteen kehitysjonosta valmiiksi saatujen asiakohtien ja aiempien sprinttien tuotos. Tuoteversion tulee olla käyttökelpoisessa kunnossa. Se täytyy olla määritelty valmiiksi riippumatta siitä, päätetäänkö se julkaista vai ei. (4.)

Scrum sisältää ennalta määrättyjä tapahtumia, joilla luodaan säännöllisyyttä ja minimoidaan tarpeet ylimääräisille kokouksille. Kaikille tapahtumille on rajattu aika, jonka Scrum voi sille sallia. Kehityssyklin alettua tapahtumiin jaettua aikaa ei saa enää muuttaa. (4.)

Scrumin ytimenä toimivat **sprintit**. Sprintti on enintään kuukauden pituinen tapahtuma, jonka sisällä pyritään luomaan mahdollinen julkaisukelpoisen tuotteen tuoteversio. Sprintin aikana ei tehdä muutoksia, jotka voivat vaarantaa sprintin tapahtumat tai laskisivat laadun tavoitteita. Kutakin sprinttiä voidaan katsoa enintään kuukauden pituisina projekteina, joiden tavoitteena on saada jotain ai-

kaiseksi. Uusi sprintti alkaa heti vanhan sprintin päätyttyä. Sprintti koostuu kaikista kehitykseen liittyvistä tapahtumista, joita ovat sprintin suunnittelu, päivittäiset scrumit, kehitystyö, sprintin katsaus ja sprintin retrospektiivi. (4.)

Sprintin **suunnittelu** toteutetaan koko projektiryhmän tai toisin sanoen scrumtiimin voimin. Tavoitteena on arvioida, mitä saadaan toteutettua seuraavan sprintin aikana ja kuinka se tullaan toteuttamaan. Suunnittelu on rajoitettu enintään kahdeksan tunnin pituiseksi kuukauden pituista sprinttiä kohden. Ryhmä pyrkii ennustamaan, mitä tuotteen kehitysjonon asiakohtia saadaan valmiiksi, keskustellen samalla, mikä on tämän sprintin tavoite. Ainoastaan projektiryhmällä on valtuudet päättää sprinttiin sisältyvistä asiakohdista. (4.)

Päivittäiset scrumit ovat lyhyitä päivittäisiä palavereja, joissa ryhmä käy läpi edellisenä päivänä aikaansaadut tuotokset ja sen, miten he aikovat seuraavaksi edistää projektin kehitystä. Samalla mietitään, onko mahdollisesti jotain, joka estää sprinttiä saavuttamasta tavoitteitaan. Päivittäisen scrumin pituus pyritään pitämään 15 minuutissa. Päivittäisiä scrumeja käytetään tavoitteiden saavuttamisen edistyksen tarkkailuun tuotteen kehitysjonon tilanteeseen verrattuna. Sillä käytännössä optimoidaan todennäköisyys, että ryhmä saavuttaa sprintin tavoitteet. (4.)

Kehitystyö on kutakuinkin itsensä selittävä. Scrumin kannalta tulee kuitenkin tietää, että projektiryhmän jäsenet ovat tasavertaisia ja kukin jäsen vastaa itse tuotoksestaan. Kehitystyö täyttää kaiken ajan, jota muihin tapahtumiin ei käytetä.

Sprintin katsaus pidetään sprintin lopuksi. Siinä tutkitaan aikaansaatua tuoteversiota ja tuotteen kehitysjonoa sovelletaan tarvittaessa. Tässä vaiheessa projektiryhmä arvioi aikaansaannoksen projektin tilaajan kanssa. Aikaansaannosten ja mahdollisten muutosten perusteella kaikki osalliset miettivät, mitä tulisi seuraavassa sprintissä saada aikaan. Sprintin katsaus on yleensä rajattu neljän tunnin pituiseksi kuukauden pituista sprinttiä kohden. (4.)

Retrospektiivi eli eräänlainen arviointitilaisuus toteutetaan katsauksen jälkeen ja ennen seuraavan sprintin suunnittelua. Retrospektiiviin osallistuvat vain projektiryhmän jäsenet. Sprintin retrospektiivi antaa projektiryhmälle mahdollisuuden tutkia itseään ja suunnitella parannuksia seuraavaan sprinttiin. Tähän on yleensä varattu kolme tuntia aikaa kuukautta kohden. Retrospektiivin tarkoitus on arvioida edellisen sprintin sujumista henkilöstön, prosessien ja työkalujen kannalta, tunnistaa potentiaaliset parannukset ja suunnitella parannusten soveltaminen ryhmän työskentelyyn. (4.)

2.2 Tuotannon hallinta

Siinä missä ohjelmistotuotantomallit antavat menetelmät tuotannon järjestelyyn, tuotannon hallinta antaa työkalut näiden menetelmien ylläpitoon. Omin sanoin kuvattuna tuotannon hallinta määrittelee työkalut ja säännöt, joiden puitteissa projekti toteutetaan.

2.2.1 Tehtävien ja ajanhallinta

Projektia aloitettaessa tulisi arvioida projektiin kuluva aika. Hyvänä tapana on laskea arvioitu aika ja lisätä siihen 50 %. Ensimmäisiä projekteja luodessa ei arvioidun ajan kannata ylittää kolmea kuukautta. (6.) Näin voidaan olettaa yrityksen saavan jotain valmiiksi ja kasvattavan resurssejaan edes jonkinmoisella rahavirralla, jos tarkoituksena on julkaista tuote ennen sen valmistumista. Projektien laajuuden tulisi elää yrityksen resurssien mukaan.

Ennen tehtävän aloittamista on hyvä kirjoittaa ylös, mitä on edellisellä viikolla saanut aikaan ja mitä seuraavalla viikolla on suunnitteilla tehdä (6). Näitä vertaillessa alkaa muodostumaan käsitys erilaisiin tehtäviin kuluvasta ajankäytöstä.

Vaikka projektia tehtäisiin yksin, on hyvä selvittää, mitä tehdä ja milloin. Tätä varten on kehitetty erilaisia seurantatyökaluja ja -malleja. Internetistä löytyy lukemattomia valmiita työkaluja kutakin ohjelmistotuotantomallia kohtaan. Näillä työkaluilla on taas erilaisia käyttötapoja. Toiset vaativat oman palvelimen, jolle

perustaa hallinnan, ja toiset taas tarjoavat sen pientä korvausta vastaan. Yksinkertaisimmillaan seurantatyökalu voi olla vaikka Excel-taulukko (kuva 3). Tärkeintä on, että projektiryhmän tekemisissä pysytään ajan tasalla.

Ids of Sprint I	Task ID		Workingdays in Sprint	29	30	1	2	3	6	7	8	9	10	13	14										
		estimated (h) per task	Description of the tasks:	team hours per working day																					Σ
1			**Toimiva testikartta**																						
	1.1	1	Testipohja		1																				1
	1.2	1	Esteitä		1																				1
	1.3	1	Navigaatio layer		1																				1
2			**Vuoropohjainen toiminen**																						0
	2.1	1	Vuoron vaihto									2	4												6
	2.2	1	Vuoron listaus		1		1																		2
	2.3	8	Toimintojen toteuttaminen							1			2												3
	2.4	2	Toimintojen jonottaminen																						0
3			**Yksikön hyökkääminen**																						0
	3.1	8	Hyökkäys RNG																						0
	3.2	3	Hyökkäyksen toteuttaminen																						0
																									0
4			**Yksikön liikkuminen**																						0
	4.1	1	Koordinaatteihin navigoiminen			5								2											7
	4.2	4	Esteiden väistäminen (muodostelma)											1											1
	4.3	3	Kiintopisteiden välillä navigoiminen																						0
	4.4	12	Muodotelman preview																						0
5			**Testitekoäly**																						0
	5.1	2	Kohteeseen liikkuminen				2		1																3
	5.2	16	Kohteeseen hyökkääminen						1	2															3
	5.3	2	Vuoron vaihto										1												1
	5.4	16	Kohteen valinta		1								1												2
6			**Taistelun lopettaminen**																						0
	6.1	4	Tavoite (vastustajan tuhoaminen)																						0
																									0
																									0
7			**Yksikön statistiikka**																						0
	7.1	2	Oletus statistiikka			2	1																		3
	7.2	4	Perusvaikutus (liikkumis, hyökkäys)				2																		2
	7.3	12	Varusteiden vaikutus (simple)																						0
																									0
8			**Yksikön muodostelma**																						0
	8.1	2	"häröpallo"																						0
	8.2	8	Perusmuodostelma (suorakulmio)													1									1
9			**Yksikkötyyppi**																						0
	9.1	16	Varustuksen vaikutus																						0
	9.2	8	Yksikön rakenne																						0
	9.3	4	Yksilön luonti																						0
10			**Yksikön rakenne**																						0
	10.1	8	Johtaja(ES)			1	1																		2
	10.2	2	Jäsen									2		2											4
	10.3	4	Jäsenen paikka											2	1										3
	10.4	4	Line of sight							4	4														
				0	5	8	7	0	2	7	4	4	8	7	2	0	0	0	0	0	0	0	0	0	54
Task Status																									
			Workload left	160	155	147	140	140	138	131	127	123	115	108	106	106	106	106	106	106	106	106	106	106	

KUVA 3. Scrumin sprintin tehtävälistan seurantatyökalu Excelissä.

Seurantatyökalujen suurin etu löytyy tehtävien jaosta projektiryhmien kasvaessa. Seurantatyökaluilla voidaan projektin elinkaari jakaa kehityssykleihin. Kehityssyklien kanssa voidaan helpottaa projektin pituuden selvittämistä ja rajata kehitys niin, että valmiit ominaisuudet luovat pelattavan paketin jo, ennen kuin peli on valmis. Tämä on hyvin tärkeä asia, jos yrityksen resurssit ovat vähäiset ja yritys haluaa julkaista jotain mahdollisimman nopeaa. Jos tarkoituksena on julkaista tuote etukäteen, tulee kuitenkin muistaa, että hätiköity projekti aiheuttaa usein enemmän haittaa kuin hyvää.

2.2.2 Versionhallinta

Versionhallinnalla tarkoitetaan järjestelmää, joka pitää kirjaa muutoksista ohjelmaan ajan kuluessa siten, että voidaan siirtyä vanhempiin versioihin myöhemmin. Versionhallinta on ehkä koko projektin tuotannon tärkein osa-alue. Versionhallinta mahdollistaa ominaisuuksien kehityksen erikseen samalla pitäen viimeisimmän toimivan julkaisun julkaisuvalmiina. Versionhallinta toimii myös eräänlaisena puskurina ominaisuuksille ja estää viallisen projektin julkaisun. Tämän takia versionhallinta on välttämätön onnistuneelle projektille. Tuotteen kehityksen aikaisen versionhallinnan tehtävä on taata kehittäjälle stabiili työympäristö, jossa projektin työntekijät eivät vahingossa häiritse toistensa työskentelyä, esimerkiksi korjaamalla samanaikaisesti samaa moduulia kertomatta siitä muille projektiryhmän jäsenille (1).

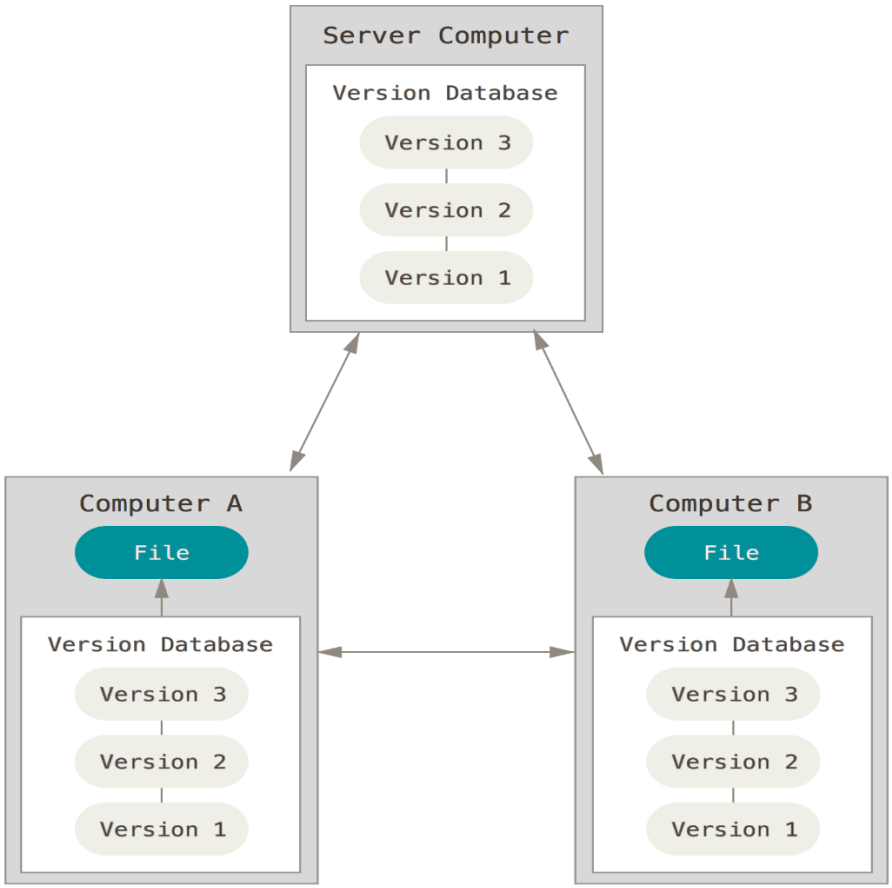
Yksinkertaisimmillaan versionhallinta on tiedoston kopioimista erilliseen kansioon. Tämä on kuitenkin hyvin altis virheille. Esimerkiksi voidaan unohtaa missä kansiossa työskentelet, ja tiedosto kopioidaan vahingossa väärän tiedoston päälle. Tämän ongelman ratkaisemiseksi ohjelmoijat kehittivät kauan sitten paikallisen versionhallintajärjestelmän, jossa on yksinkertainen tietokanta, joka säilyttää kaikki muutokset muutoksenhallintaan. (7.)

Seuraava iso ongelma oli, että ihmisten täytyi työskennellä toisten kehittäjien kanssa eri järjestelmissä. Tähän ongelmaan kehitettiin keskitetty versionhallintajärjestelmä. Nämä järjestelmät kuten CVS, Subversion ja Preforce perustuvat yhteen palvelimeen, joka sisältää kaikki versioidut tiedostot ja käyttäjät, jotka muokkaavat tiedostoja tästä keskitetystä paikasta. (7.)

Tällä järjestelmällä on monta vahvuutta, erityisesti paikallisiin versionhallintajärjestelmiin verrattuna. Esimerkiksi kaikki tietävät kutakuinkin, mitä kukin tekee, ja järjestelmän haltijat hallitsevat, kuka voi tehdä ja mitä. Tämän lisäksi on paljon helpompaa hallita keskitettyä järjestelmää kuin jokaisen käyttäjän paikallista järjestelmää. (7.)

Tällä järjestelmällä on kuitenkin huonotkin puolensa. Kaikista itsestäänselvin on se, mitä keskitetty palvelin edustaa. Esimerkiksi jos palvelin kaatuu tunniksi, ei kukaan kehittäjistä voi tehdä yhteistyötä tai pääse käsiksi tiedostojen muuttamiseen tämän tunnin aikana. Jos palvelin korruptoituu ja varmuuskopioita ei ole tehty, menettivät käyttäjät aivan kaiken. Tämä on suurin keskitettyjen järjestelmien riski. (7.)

Tässä hajautetut versionhallintajärjestelmät ovat eduksi. Hajautetuissa järjestelmissä, kuten Git, Mercurial, Bazaar tai Darcs, käyttäjät eivät pelkästään ota viimeisintä tilannekuvaa tiedostoista, vaan he peilaavat koko säilytyspaikan omalle järjestelmälleen. Näin ollen jos palvelin kaatuu, voi kuka tahansa käyttäjistä palauttaa tiedostot järjestelmään omasta järjestelmästään. Käytännössä kullakin käyttäjällä on varmuuskopio palvelimen tiedostoista. Kuvassa 4 esitellään hajautetun versionhallintajärjestelmän rakenne. (7.)



KUVA 4. Hajautettu versionhallintajärjestelmä (7)

2.2.3 Koodin ulkoasu

Jokaisella ohjelmoijalla on oma tyylinsä tuottaa koodia. On kuitenkin suotavaa, että projekti, jota työstää useampi henkilö, olisi ulkonäöltään yhtenäinen. Kun ulkoasu noudattaa sille asetettua kaavaa, voi kuka tahansa projektiryhmästä käsitellä muiden tekemää työtä. Tästä on huima etu, jos komponentin alkuperäinen tekijä on kiinni muissa töissä ja komponentti vaatii muutoksia.

Ihan yksityiskohtaisesti en ota tässä osiossa aiheeseen kantaa, mutta koodin ulkoasu on kuitenkin keskeinen osa toimivassa projektiryhmässä. Käytin lähteenä Microsoftin Code Complete -kirjan toista painosta, joka kertoo hyvin kattavasti yleisestä ohjelmoinnista (8). Kirja ei puutu mihinkään yksittäiseen ympäristöön tai kieleen, vaan se kertoo vihjeitä ja näkökulmia, joita voi hyödyntää kaikkialla. Siksi koin kirjan hyväksi lähteeksi työlleni. Kirjan kirjoittaja väittää kirjan olevan

ainoa maailmassa, joka lähestyy ohjelmointia näin käytännöllisestä näkökulmasta.

Kuka tahansa vähänkin ohjelmointia osaava osaa kirjoittaa koodia, jonka tietokone ymmärtää, mutta projektiryhmässä työskentely vaatii koodilta rakenteen, jota ihminen ymmärtää. Ideana onkin, että ohjelmakoodia kirjoitetaan toiselle ihmiselle eikä tietokoneelle. Vaikka työskenneltäisiinkin yksin, on ohjelman ulkoasulla merkitystä, koska usein joutuu palaamaan vanhan koodin pariin parantelun tai muutoksien merkeissä. Voidaan vain kuvitella, kuinka työt helpottuvat, kun ohjelmakoodi on helppolukuista.

Ohjelmakoodin hyvältä näyttäminen on toki tärkeä asia, mutta se ei ole kuitenkaan tärkein asia koodia kirjoittaessa. Koodin ulkoasussa tärkein asia on, että koodin rakenne on selvä. Jos toinen koodi näyttää paremmalle, mutta toisesta tulee rakenne paremmin selville, käytetään sitä, joka tuo rakenteen paremmin selville. Ideana on rakentaa koodi loogiseen järjestykseen. Periaatteessa loogisesti rakennettu koodi on hyvän näköinen, ellei logiikka itse ole rumaa jälkeä. Näin ollen logiikka mielessä rakennettu koodi kertoo ohjelman laadun, kun taas ulkoasu mielessä rakennettu koodi piilottaa sen. (8, s. 732.)

Kun ohjelmakoodia kirjoitetaan ihmiselle, mukaan voidaan ottaa tyhjiä rivejä, sisennyksiä ja ylimääräisiä välilyöntejä, joita tietokone ei ota huomioon. Nämä eivät vaikuta ohjelman tehokkuuteen mitenkään, eli näitä lisäämällä ei hävitä mitään. Sen sijaan kirjoitettaessa ohjelmakoodia ihmiselle voidaan ulkoasulla tuoda ilmi idea ohjelman rakenteesta. Esimerkiksi virhetapauksissa, jos koodista puuttuu sulkumerkki, voidaan sisennyksillä kertoa korjaajalle, mihin paikkaan sulku oli tarkoitus laittaa.

Koodia kirjoittaessa tulee muistaa, ettei ole yhtä ainoaa ratkaisua oikeaan ulkoasuun. Näin ollen ryhmässä työskennellessä tulisi ennen työhön ryhtymistä keskustella käytettävästä ulkoasusta, koska luettava ulkoasu tulee tuottaa jo koodia kirjoitettaessa eikä sen jälkeen.

Hyvän koodipohjan tavoitteena on luoda ohjelmakoodi esittämään tarkasti koodin loogista rakennetta. Tyypillisesti ohjelmoijat toteuttavat tämän sisennyksillä ja tyhjillä tiloilla. Toisaalta loogisesti rakennettu koodi, jota on vaikea lukea, on käyttökelvoton. Hyvä koodipohja panostaa luettavuuteen. Ohjelmakoodin tulisi myös kestää muutokset. Yhden rivin muuttamisen ei tulisi vaikuttaa muihin. (8, s. 735.)

Tyhjät tilat, kuten välilyönnit, tabuloinnit, rivin vaihdot ja tyhjät rivit, ovat ensisijainen työkalu ohjelman rakenteen kuvaamiseksi. Esimerkiksi kirja, jossa ei käytetä välilyöntejä ollenkaan, on hyvin vaikealukuinen. Ohjelmakoodi noudattaa samaa periaatetta. Käyttämäni lähde keskittyi kolmeen ydinasiaan selkeälukuisen ohjelmakoodin tuottamisessa.

Ensimmäisenä on koodin **ryhmittäminen omiksi kappaleiksi**. Kirjoitettaessa kappaleet sisältävät lauseita, jotka liittyvät johonkin ajatukseen. Samalla tavalla koodia kirjoitettaessa kappaleen tulisi sisältää lausuntoja, jotka toteuttavat yhden tehtävän ja liittyvät toisiinsa. Toisin kuin kirjaa kirjoittaessa, lauseet tulisi erottaa rivinvaihdolla. Liian pitkä rivi koodia voi käydä hankalaksi lukea. (8, s. 737.)

Toisena on **toisiinsa liittymättömien rivien erottaminen toisistaan**. On aivan yhtä tärkeää erottaa koodia toisistaan kuin yhdistää sitä. Tyhjän rivin jättäminen kappaleiden väliin on tässä oiva keino ohjelman selkeyttämiseksi. Tyhjää riviä käytetään usein rutiinien erottamiseen toisistaan ja kommenttien korostamiseen. (8, s. 737.)

Kolmantena ydinasiana ovat **sisennykset**. Itse koen tämän ehkä tärkeimmäksi ulkoasun tekijäksi koodia kirjoitettaessa. Sisennyksillä ilmaistaan koodin looginen rakenne. Sääntönä lauseet, jotka ovat edellä olevan rivin alaisuudessa, tulisi sisentää. (8, s. 737.)

Tyylejä käyttää näitä ydinasioita on lukuisia. Vaikka edellä mainitut tyylit korostavat koodin luettavuutta, ei kuitenkaan näiden kanssa tule liioitella. Liialliset si-

sennykset voivat aiheuttaa epäselvyyttä, ja ylimääräiset tyhjät rivit lisäävät ohjelman pituutta turhaan. Kaiken lisäksi erään tutkimuksen mukaan optimaalinen tyhjien rivien määrä koodissa on 8–16 prosenttia. Jos niitä on tätä enemmän, ohjelman debuggaukseen eli virheiden etsintään kulunut aika kasvaa dramaattisesti. (8, s. 737.)

3 TUOTANTOPROSESSI PELIALALLA

Pelialalla kokonaistuotantoprosessissa on hieman erilainen lähestymistapa kuin tavanomaisessa ohjelmistotuotantoprosessissa, mutta pohjimmiltaan ohjelmistotuotantoperiaate on sama. Peliala koetaan luovaksi alaksi eli se sisältää muutakin kuin ohjelmointia, ja projektin vaatimukset tulevat usein kehittäjien vision pohjalta, jos kyseessä ei ole lisensoitu konsepti, esimerkiksi Disney-elokuviin pohjautuva peli. Pelialla yleisiin tuotannon vaiheisiin viitataan usein termein konsepti, esituotanto, tuotanto ja jälkituotanto.

3.1 Konsepti

Konsepti vastaa ohjelmistotuotannon esitutkimusvaihetta. Pelin tuotanto alkaa konseptista, jossa pelin sisältö käydään ideatasolla läpi. Tässä vaiheessa päätetään projektin yleinen teema, esimerkiksi onko se räiskintä- tai rallipeli. Konseptin sisältöä voidaan kuvata otsikkotasolla esittely, tausta, kuvaus, avainominaisuudet, tyylilaji, alusta ja konseptitaide. Konseptin tarkoitus on tuoda sen lukijalle kuva valmiista pelistä, ja sitä voidaan käyttää pelin alustavana markkinointityökaluna.

Konsepti alkaa **idean esittelystä,** jossa tavoitteena on esittää pelin idea yhdellä lauseella lukijalleen mahdollisimman houkuttelevasti. Tämän lauseen tulisi sisältää pelin nimi, tyylilaji, suunta, alusta, tausta ja mitä tahansa muuta tietoa, joka tulisi kertoa lukijalle heti. Esittelyn jakaminen useampaan lauseeseen on suotavaa, mutta mitä kauemmin esittely kestää, sitä laimeammalta idea vaikuttaa. (9.)

Pelin taustalla tarkoitetaan laajennuksia, joiden päälle peli on rakennettu. Siinä kuvataan esimerkiksi, onko peli jatko-osa toiselle pelille tai käyttääkö peli lisensoituja ohjelmia. Näin ollen tämä osio ei ole välttämätön konseptille ja tulisi pitää erillään esittelystä, että lukija voisi halutessaan välttää lukemasta sitä, jos hänellä on jo tarpeellinen tieto käsillä. Tausta on kuitenkin tärkeä, jos peli käyttää

lisensoituja ohjelmia tai aiempaa materiaalia. Jos kehittäjällä on aikomus käyttää jo olemassa olevia työkaluja, tulisi ne kuvata tässä osiossa. (9.)

Kuvauksella pyritään parilla kappaleella tai sivulla kuvaamaan lukijalle peli, kuin he olisivat sen pelaajia. Kuvaus kerrotaan sinä-muodossa pelaajan näkökulmasta katsottuna. Tarkoituksena on kirjoittaa tämä jännittäväksi selostukseksi pelaajan kokemuksista. Kuvauksen tulisi sisältää lähes täsmällinen selostus pelaajan tekemisistä, mutta yksityiskohtia, kuten napin painalluksia, tulisi välttää. Kuvauksen tarkkuus tulisi olla juuri käyttöliittymän ulkopuolella. Kuvauksen tulisi kuvata konsepti- ja viihdearvo selvästi ja vakuuttavasti. (9.)

Avainominaisuudet sisältävät listan ominaisuuksista, jotka erottavat tämän pelin toisista ja asettavat tavoitteet, joita myöhemmät dokumentit ja tuotokset tavoittelevat. Ominaisuudet tuottavat yhteenvedon kuvauksessa vihjatusta pelikokemuksesta. Nämä ominaisuudet toimivat työkaluna pelin markkinoinnille, ja niiden tulisi sisältää selvitys kunkin ominaisuuden sisällöstä. Kun kyseessä on avainominaisuudet, ei aivan kaikkea kannata listata. Liian monta ominaisuutta lieventävät konseptin vahvimmat ominaisuudet, ja liian vähän ominaisuuksia voi antaa turhan supistetun kuvan konseptista. Tulee pitää mielessä, että selkeät ominaisuudet, kuten hienot grafiikat tai kiehtova musiikki, tulisi pitää pois listasta, ellei sitten kyseessä ole huomattavasti paremmat ominaisuudet kuin kilpailijoilla. Toisaalta, jos pelissä tulee olemaan omalaatuiset grafiikat tai äänet, jotka erottavat tuotteen markkinoilla, tulisi niitä mainostaa. (9.)

Tyylilajilla kuvataan parilla sanalla pelin pelimekaaninen suuntautuminen. Tyylilajia kuvatessa tulisi käyttää jo laajalti tunnettuja ja käytettyjä termejä, kuten first-person shooter tai real-time strategy. Tätä voidaan vielä laajentaa niche-markkinoille eli kapeammin määritellyn kohderyhmän markkinoille, esimerkiksi sanoilla fantasy, modern tai sci-fi. (9.)

Alustalla kuvataan kohdealusta(t), jolle peliä ollaan kehittämässä. Jos konseptin koetaan sopivan useammalle alustalle, tulisi ne kaikki kuvata tässä vaiheessa ja myös kertoa, mille alustalle peliä suositaan. Jos tavoitteena on tehdä moninpelituettu peli, voidaan sekin kertoa tässä vaiheessa. (9.)

Konseptitaide ei ole välttämätön konseptille, mutta pienikin näyte pelin ulkoasusta voi myydä idean lukijoille. Taidetta voidaan käyttää pelin uniikkien tai monimutkaisten ideoiden esille tuomiseen. Taiteella voidaan kuvata pelimaailman ympäristöä tai hahmoja, jotka asettavat lukijan samoille aallonpituuksille kehittäjien kanssa. (9.)

3.2 Esituotanto

Esituotanto vastaa ohjelmistotuotannon suunnitteluvaihetta. Tässä vaiheessa toteutetaan dokumentit pelin ominaisuuksista ja kartoitetaan projektin laajuus. Ohjelmistotuotannon ulkopuolella tähän vaiheeseen otetaan mukaan pelin juonen kehitys ja peliympäristön kehitys. Tavoitteena on tuottaa pelille kattava pohjapiirustus.

Esituotannossa projektiryhmän rajoitteet liittyvät usein siihen, minkä tyylistä peliä ollaan kehittämässä. Jos peliä kehitetään täysin alkuperäisen konseptin pohjalta, ovat suunnitteluvaiheessa vain kehittäjien luovuus ja teknologiset rajoitteet rajana. Jos peliä kehitetään lisensoidulle pohjalle, kuten elokuvalle, on kehitys yleensä rajoitettu lisenssin haltijan määrittelemien rajojen sisälle. Esimerkiksi hahmot eivät saa tehdä tai sanoa mitään, jota he eivät tekisi elokuvassa. (10.)

3.3 Tuotanto

Pelialalla tuotanto vastaa tavanomaisen ohjelmistotuotannon malleja. Pelialalla käytetyimmät vaihejakomallit perustuvat ketteriin menetelmiin niiden nopean reagointikyvyn johdosta. Tässä vaiheessa aletaan toteuttamaan peliä esituotannossa saatujen dokumenttien pohjalta. Viimeistään nyt tulevat esille puutteet, jotka jäivät suunnittelussa ennalta näkemättä. Jos puutteita tulee, on kehittäjien pakko löytää ratkaisut ongelmiin.

Ensimmäisten tuotannon tehtävien tavoitteena tulisi olla pienin mahdollinen toiminnollisuus pelattavan pelin luomiseksi (6). Pienin mahdollinen toiminnollisuus käytännössä tarkoittaa sitä, että peliin kehitetään vain välttämätön ydinmekaniikka projektin testausta varten. Esimerkiksi, jos pelin ideana on tasohyppely, ydintoiminnot saattavat olla vain juokseminen, hyppy ja alusta, jolla liikkua. Kaikki muu on vain lisää ydinmekaniikan päälle ja tulisi jättää jatkokehityksen tehtäväksi.

Pienimmän mahdollisen toiminnollisuuden jälkeen selviää usein se, onko peli kannattava jatkokehitykselle. Jos peli ei tunnu tässä vaiheessa hyvältä, tulisi palata takaisin suunnittelupöydän ääreen. Jos pelin ydinmekaniikka ei miellytä, voi koko peli tuntua ikävältä, vaikka mielessä olisi useita mielenkiintoisia ominaisuuksia, joita lisätä peliin.

Jos peli tuntuu kannattavalta jatkokehitystä varten, voidaan kehittäminen aloittaa jo valmiin ydinmekaniikan ympärille. Tästä eteenpäin voidaan tehtävät jakaa erilaisiin ominaisuuksiin pelin syvyyden parantamiseksi tai edellisten ominaisuuksien korjaamiseen.

3.4 Jälkituotanto

Laadunhallinta on hyvin oleellinen osa pelin kehitystä. Peleillä ei ole varsinaisia laatustandardeja, vaan yritys määrittää itse haluamansa kriteerit pelin laadulle. Jos peliä valmistaa konsoleille, tulee kehittäjien tuottaa peli vastaamaan konsolin tuottajan standardeja. Esimerkiksi Microsoftin Xbox-konsolin ohjaimen toiminnoista A-painikkeen tulisi aina valikossa liikkua eteenpäin ”Advance” ja B-painikkeen taaksepäin ”Back” (10).

Ohjelmistotuotannossa on tapana julkaista pelistä niin sanottu testiversio, joka julkaistaan suljetulle yleisölle. Tämän yleisön annetaan kokeilla tuotetta, ja he sitten ilmoittavat löytämänsä virheet tuotteesta ja antavat palautetta sen toimivuudesta.

Testiversiot on tapana jakaa eri vaiheisiin. Ensimmäinen vaihe eli alfaversio julkaistaan hyvin suppealle yleisölle, ja sen tarkoituksena on hioa pelin ydinmekaniikka toimivaan muotoon. Vielä tässä vaiheessa sallitaan erilaisia puutteita. Seuraava vaihe eli beetaversio julkaistaan suuremmalle suljetulle yleisölle, joka koostuu potentiaalisista asiakkaista. Tässä vaiheessa tavoitteena on korjata kaikki löydetyt viat ja hioa peli julkaisukuntoon. (1.)

Yleensä beetasta jää jälkeen vielä muutamia vikoja, jotka ovat jääneet testaajilta huomaamatta. Nämä viat yritetään korjata mahdollisimman nopeasti julkaisun jälkeisillä päivityksillä. Nämä päivitykset jatkuvat projektin elinkaaren loppuun asti. Vaikka kaikki viat saataisiinkin korjattua, voi lisäpäivityksille tulla tarvetta laitekokoonpanon kehittyessä.

4 OMAN YRITYKSEN TUOTANTOPROSESSI

Työn lopputulokseksi sain prosessimallin, joka oli hyvin lähellä pelialalla yleisesti käytettyä mallia. Tästä huolimatta työ avasi minulle, miten asia tulisi käytännössä hoitaa.

4.1 Lähtökohta

Ennen opinnäytetyön kirjoittamista olin kokeillut pelin kehittämistä useampaan otteeseen. Ensimmäisillä kerroilla yritin kehittää peliä suoraan idean pohjalta, ja ne yritykset kaatuivat nopeasti projektin laajuuden hallitsemattomaan kasvuun ja epäselvyyteen. Huomasin nopeasti ongelman kiteytyvän projektinhallinnan puutteeseen.

Seuraavilla yrityksillä karsin projektin laajuutta sopimaan enemmän aloittavan kehittäjän mittasuhteisiin ja minulla oli käytössä projektin hallintaan tyydyttävät työkalut. Esimerkiksi versionhallinta tehtiin Gitin kautta ja tehtävien hallinta Trellon avulla, joka mahdollistaa tehtävien listauksen, mutta ajanhallintaa se ei tarjoa. Tämä olisi voinut riittää, jos projekti olisi mitoitettu oikein, mutta kovin järjestelmällistä kehitystä ei näillä saatu vielä aikaan. Nämä yritykset kaatuivat turhan kunnianhimoisiin hankkeisiin.

Viimeisillä kerroilla otin käyttööni ohjelmistotuotantomalli Scrumin aikaisemmin käytettyjen työkalujen lisäksi. Scrumin kanssa tuotanto sujui kohtalaisen mallikkaasti ensi alkuun. Sen käytössä oli kuitenkin hankaluuksia, koska ryhmän koko oli pieni ja dokumentoinnin ylläpitoon ei ollut tarpeeksi kokemusta eikä työvoimaa. Viimeisin yritykseni kehittää peliä kaatui lopuksi käsistä riistäytyneiden muutostarpeiden raskauteen. Tässä vaiheessa koin tarpeelliseksi kartoittaa yritykselleni sopivan tuotantoprosessin.

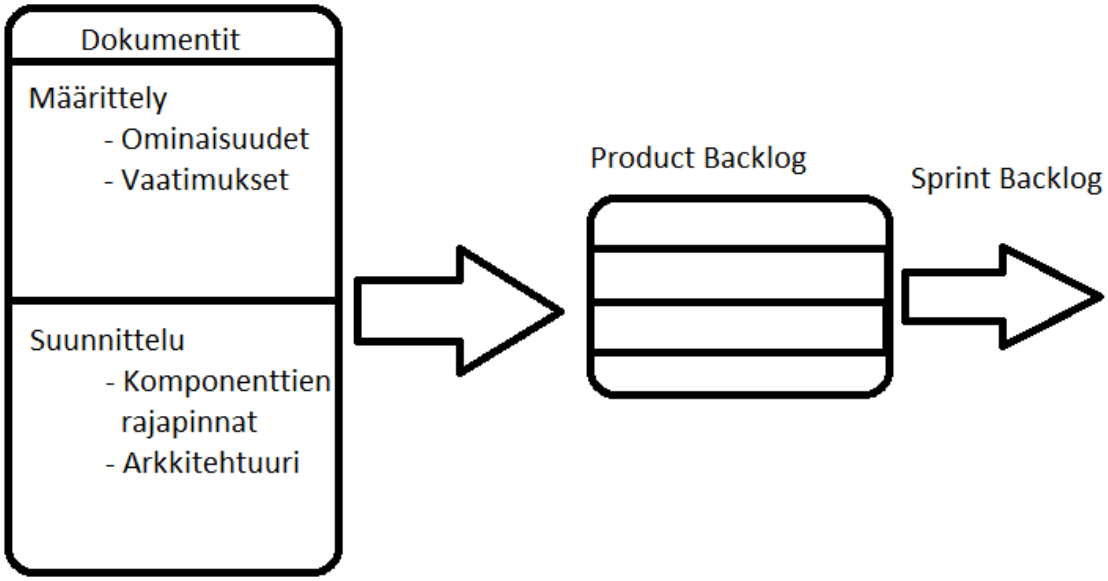
4.2 Ratkaisu

Koska viimeisin yrityskään ei onnistunut ohjelmistotuotannon mallin Scrumin kanssa raskaiden muutostarpeiden takia, koin tarpeelliseksi suunnitella projektilleni kattavan rungon, jonka päälle rakentaa. Ketterät menetelmät koin vielä tarpeelliseksi, sillä huolimatta perusteellisesta suunnittelusta tulee aina olemaan tarvetta muutoksille, ja nämä menetelmät mahdollistavat nopean reagoinnin muutoksiin. Näin ollen päädyin tulokseen, että tarvitsen kattavan dokumentoinnin projektin rungosta ennen tuotannon aloittamista. Dokumentoinnin valmistuttua perustan Scrumin tuotteen kehitysjonon suunnitelmien pohjalta ja tuotannon hoidan normaalilla Scrumin käytännöllä. Scrumin ylläpitoon sovelsin koulusta saamaani Excel-pohjaa (liite 1). Hallussani ei ole vielä toimitiloja, joten Scrumin suosimat päivittäiset palaverit saattavat jäädä ryhmältäni pois. Vastauksena siihen pyrin pitämään yhteyttä ryhmääni Skypen eli internetpuhelimen avulla. Tärkeintä on, että kaikki ryhmän jäsenet ovat ajan tasalla ryhmän tekemisistä.

4.2.1 Dokumentointi

Dokumentoinnin hoidan soveltamalla pelialla käytettyä konseptin mallia (9), jonka löysin, ja vesiputousmallin määrittely- ja suunnitteluvaihetta. Konsepti toimii erinomaisena lähteenä vaatimusten kartoittamiseen, ja vesiputousmallin määrittely- ja suunnitteluvaiheet voivat jopa kevennettynä tuottaa selkeät raamit projektin rakenteelle.

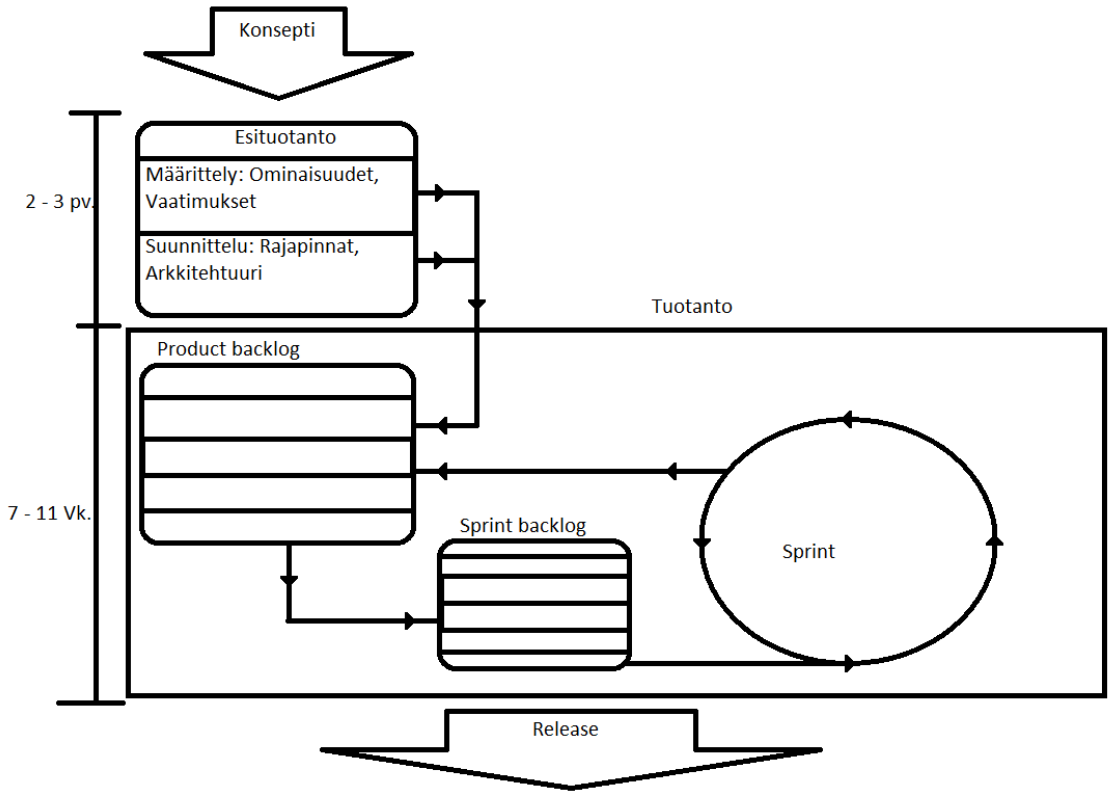
Sovellettuun määrittelyvaiheeseen kirjaan konseptin perusteella projektin vaatimukset ja ominaisuudet. Sovellettuun suunnitteluvaiheeseen kartoitan lähinnä komponenttien viestinnän ja rajapinnat. Yksityiskohtiin en suunnittelussa pyri, koska Scrum toimii parhaiten juuri vapaana tuotannon mallina. Tavoitteena on kuitenkin selkeyttää kunkin komponentin rakenne, että radikaaleilta muutoksilta vältyttäisiin. Tämä eroaa tavanomaisesta tuotantoprosessista siten, että Scrumin päälle luodaan kattava dokumentointi. Dokumenttien käyttötapa on esitetty kuvassa 5.



KUVA 5 Dokumenttien järjestys ja sisältö

4.2.2 Hallintatyökalut ja menetelmät

Ajankäytöllisesti pyrin suositeltuun enintään kolmen kuukauden projektiin. Mitoitan projektin kahden kuukauden pituiseksi, ja lisään siihen suositellun 50 % varmuuden vuoksi (6). Konseptin kirjoitan tuotannon ulkopuolella, joten en laske sitä kokonaisajankäyttöön. Muuhun dokumentointiin suunnittelin käyttäväni kahdesta kolmeen päivään. Koen sen riittävän hyvin, kun kyseessä ovat vain kevyet raportit projektin rakenteesta. Näin ollen tuotantoon ja jälkituotantoon on varattu seitsemästä yhteentoista viikkoa. Kun tuotannon mallina on Scrum, täytyy tuotanto jakaa sprintteihin. Ajallisesti näin pieneen projektiin oletan, että viikko sprinttiä kohden riittävää hyvin. Tuotantoprosessi on esitetään kuvassa 6.



KUVA 6 Tuotantoprosessikaavio

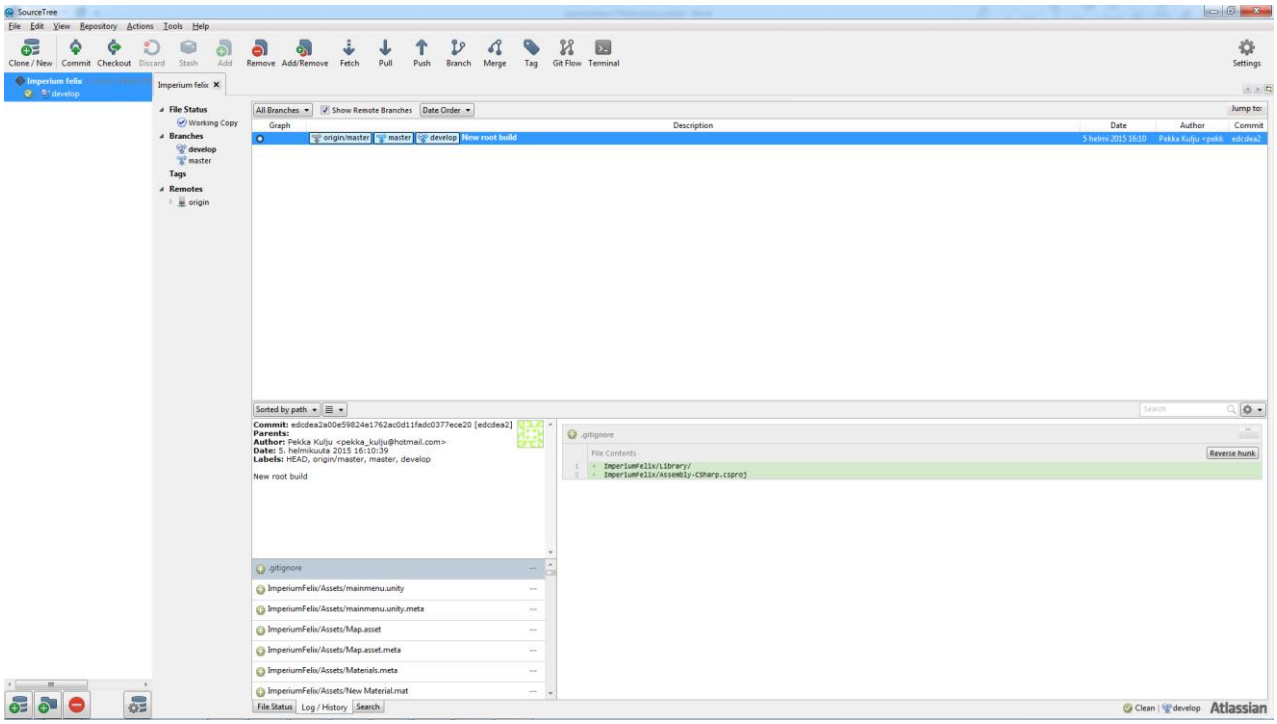
Tuotannossa tulevat rajalliset resurssit parhaiten esille. Normaalisti projektiryhmässä on omat hallintaelimet, ohjelmoijat ja muu tuotanto erikseen (4; 11), mutta minun ryhmäkoossani joudun asettamaan päällekkäisyyksiä tehtävissä. Tämä johtaa lähinnä siihen, että joudun yksinkertaistamaan projektin hallintaa kuitenkin tavoitellen pienimmän tarpeellisen ylläpidon paremmalla puolella olemista. Muista tehtävistä en voi karsia, sillä se vaikuttaisi ulospäin näkyvän tuotteen laatuun.

Käytössäni on minun lisäkseni kaksi henkilöä, joista yksi on erikoistunut grafiikkaan ja toinen ohjelmointiin. Itse sijoitun hallintatehtäviin ja ohjelmointiin. Äänen tuottajaa en ole löytänyt, joten toistaiseksi se ulkoistetaan. Jokainen ryhmän jäsen osallistuu suunnitteluun, johon myös hankitaan ulkopuolelta apua. Henkilöstön tehtävänjako on havainnollistettu taulukossa 1.

TAULUKKO 1. Yrityksen henkilöstön tehtäväjako kolmelle henkilölle

	Koko-nais-määrä	**Hal-linta**	**Ohjelmointi**	**Grafiikka**	**Ääni**	**Suunnittelu**
Sisäinen	3	1	2	1		3
Ulkois-tettu	x				x	x

Versionhallinnassa päädyin käyttämään hyvin yleistä ja suosittua Gitiä ja siihen liitettävää graafista käyttöliittymää Sourcetreetä. Sourcetreen käyttöliittymä on esitetty kuvassa 7. Sourcetree ei ole mitenkään pakollinen projektin hallintaan, mutta se helpottaa versionhallinnan käyttöä ja antaa selkeän kuvan projektin eri vaiheista ja niiden sijoittumisesta kokonaisuuteen. Gitin ylläpitoon valitsin Bit-bucket-palvelun.

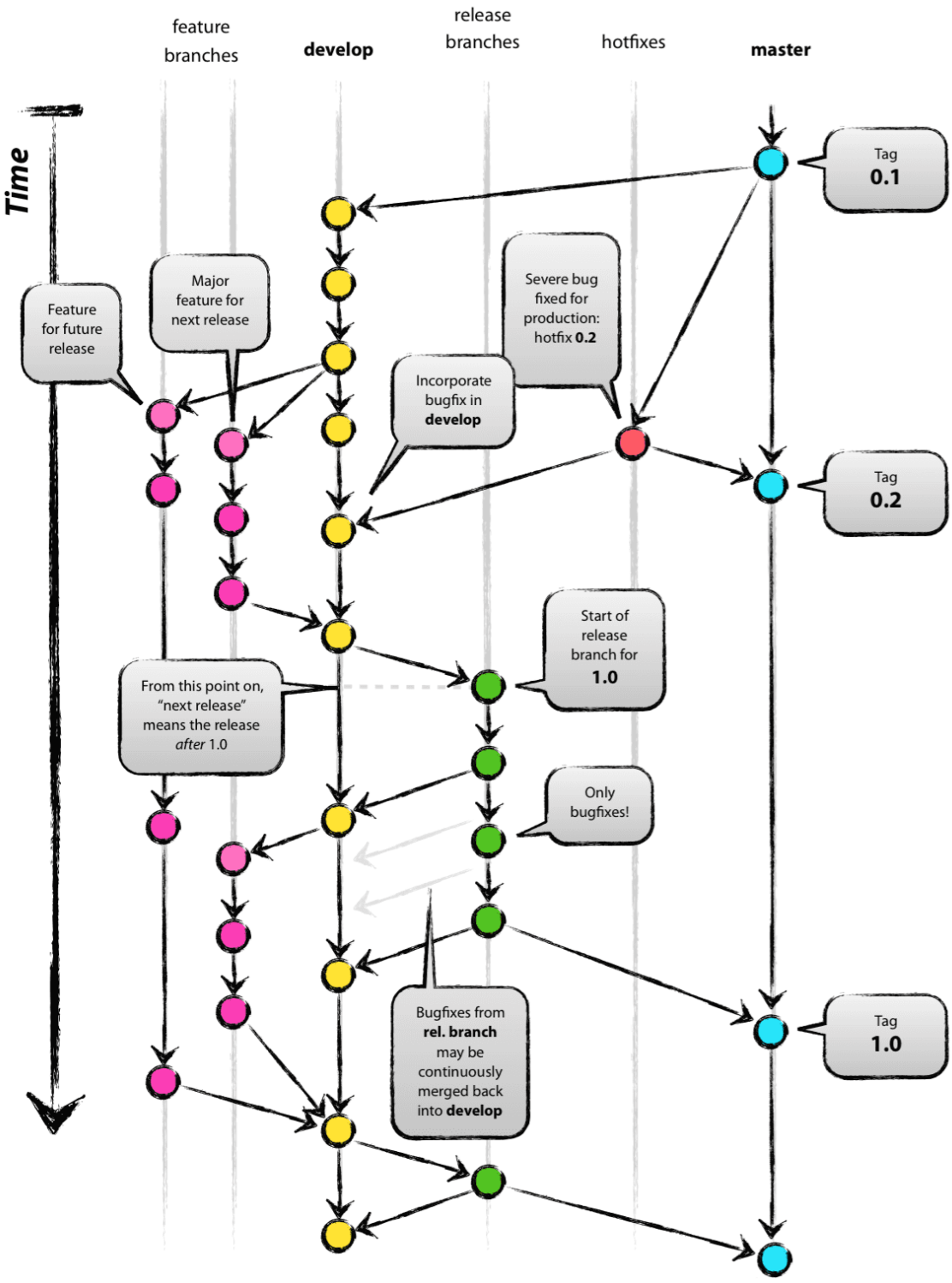


KUVA 7. Sourcetreen käyttöliittymä

Valitsin Gitin versionhallintaan, koska tarvitsin yhteistä versionhallintaa projektiryhmäni kesken ja käytössäni ei ole keskitettyyn versionhallintajärjestelmään vaadittua palvelinta. Ainoana haittapuolena voisin kuvitella tietoturvariskit, joista en ole huolissani vielä tässä vaiheessa. Versionhallinnan ylläpitoon valitsin Bitbucketin, koska se tarjoaa palvelun ilmaiseksi alle viiden henkilön ryhmille rajattomalla määrällä tallennuspaikkoja. Tämä sopii ryhmäkokoonpanolleni erittäin hyvin.

Löysin Gitin käyttöön minulle erittäin hyvin sopivan pohjan (12). Se perustuu niin sanottuun branchaukseen ja mergettämiseen tai suomeksi haaroittamiseen ja yhdistämiseen, missä kaksi ydinhaaraa (master eli päähaara ja develop eli kehityshaara) elävät koko projektin elinkaaren läpi. Päähaara sisältää aina julkaisuvalmiin tuotteen, ja sen sisällä ei tapahdu kehitystyötä. Projektin kehitystyö tapahtuu kehityshaaran sisällä, josta yhdistetään päähaaraan uusi versio, kun kehityshaara on saavuttanut stabiilin pisteen. (12.)

Kahden ydinhaaran lisäksi voidaan käyttää kolmea erilaista tukevaa haaraa. Toisin kuin ydinhaarat, nämä ovat käytössä tilapäisesti toteuttamassa niille tarkoitettua tehtävää. Ensimmäinen näistä on features eli ominaisuushaara, jonka käyttötarkoitus on kehittää yksittäisiä ominaisuuksia erillään keskeisestä kehityshaarasta. Ominaisuushaaran tulee aina lähteä kehityshaarasta, ja sen tulee aina yhdistyä siihen takaisin, kun kehitetyn ominaisuuden koetaan olevan valmis. Toinen on release eli julkaisuhaara, jonka käyttötarkoitus on tukea uuden version julkaisua. Se mahdollistaa viime hetken viilaukset ja muun datan valmistelun, kuten versioinnin. Julkaisuhaara otetaan kehityshaarasta silloin, kun kehityshaara vastaa lähestulkoon seuraavaan julkaisuun haluttua kokoonpanoa. Julkaisuhaara yhdistetään pää- ja kehityshaaraan, kun sen käsittely on valmis. Kolmantena on hotfix eli pikakorjaushaarat, joiden tarkoitus on toimia pikaisten korjausten kehitysympäristönä silloin, kun päähaaraan on päässyt huomaamaton virhe, joka täytyy korjata mahdollisimman nopeasti. Pikakorjaushaara yhdistetään korjauksen jälkeen pää- ja kehityshaaraan. Gitin haaroittamisen käytäntö on esitetty kuvassa kuva 8. (12.)



KUVA 8. Gitin käyttötapa (12)

4.2.3 Laatuvaatimukset

Pelin markkinointia miettiessäni olen löytänyt monta tapausta, joissa lupaavalta vaikuttava projekti on pilattu huonolla toteutuksella. Tästä syystä päätin asettaa tiukat kriteerit projektille, jotka perustuvat kriitikoiden väittämiin potentiaalisten asiakkaiden tottumuksiin (liite 2). Minulle sopiviin vaatimuksiin pääsin seuraamalla pelikriitikkojen kommentteja erilaisista peleistä. Vaikka resurssit kävisivätkin vähiin, on puolihuolimattomasti tehdyn projektin julkaiseminen enemmän haitallista kuin hyödyllistä.

Pelin suorituskyky on hyvin oleellinen vaatimus. Tavoitteenani on saada tuleva projektini toimimaan mahdollisimman monelle eri PC-kokoonpanolle. Kriitikot usein kommentoivat sitä, kuinka suuri fps eli frames per second pelillä on eri tilanteissa. Esimerkiksi liian alhainen fps näkyy pelaajalle kuvan pätkimisenä. Fps:llä tarkoitetaan sitä, kuinka monta kertaa kuva piirretään ruudulle sekunnissa, mutta se vaikuttaa myös siihen, kuinka hyvin peli reagoi käskyihin, sillä peli käy koodin läpi kertaalleen jokaisella piirrolla. PC-alustalla tämän hetkinen suosittu fps-arvo on 60 ja konsoleilla 30.

Pelin laatuasetukset ovat odottamattoman tärkeässä asemassa. Laatuasetuksilla varmistetaan se, että peli toimii mahdollisimman monella kokoonpanolla. Lisäksi asetuksissa voi olla mahdollisuuksia säätää peli sopivaksi mahdollisimman monille, kuten värisokeille väriasetuksia muuttamalla tai matkapahoinvoiville kuvaruudun kapeutta säätämällä. Myös painikkeet on hyvä voida mukauttaa pelaajan omiin tottumuksiin. Ohjelmiston puolelta tämä tarkoittaa sitä, että pelin asetusvalikko kehitetään mahdollisimman kattavaksi. Pelit, jotka mahdollistavat mahdollisimman monipuolisen mukautuksen, ovatkin niitä, jotka saavat parhaimman palautteen.

Pelin kannalta tärkein asia kuitenkin on se, että peli tuntuu hyvältä pelata. Pelin pelattavuus on se, joka pääasiassa houkuttaa asiakkaita. Pelattavuuteen vaikuttaa se, kuinka peli reagoi käskyihin ja miten pelin käyttöliittymä on suunniteltu.

Pelattavuutta ei usein saa heti haluttuun muotoon, vaan se vaatii jatkuvaa testausta tuotannon yhteydessä.

Lähtökohtaisesti testausprosessin toteuttavat pelin kehittäjät kehitystyön aikana. Kehittäjät kokeilevat aluksi yksittäisten ominaisuuksien toiminnollisuutta, jotta tiedetään projektin kokonaisuuden toimivan kutakuinkin halutusti. Kun seuraava kokoonpano valmistellaan uuteen tuoteversioon, projektiryhmä pelaa tätä kokoonpanoa löytääkseen siitä suurimmat ongelmat. Kun nämä ongelmat on korjattu, voidaan kokoonpano siirtää seuraavaan julkaisuun.

Projektin saapuessa alfaversioon aletaan projektin vikoja korjata suuremman yrityksen ulkopuolelta värvätyn testausryhmän voimin. Testausryhmä koostuu tässä vaiheessa yleensä luotetuista projektiryhmälle tutuista henkilöistä. Alfaversioon päästään silloin, kun kaikki projektin halutut ominaisuudet ovat olemassa, mutta niitä ei ole vielä hiottu. Alfaversio koetaan valmiiksi silloin, kun kaikki ominaisuudet toimivat halutulla tavalla ja peliä rikkovat ongelmat on ratkottu.

Seuraavana on vuorossa beetaversio. Beetaversiossa projekti avataan yhä suuremmalle yleisölle, ja tässä vaiheessa otetaan markkinointi suuremmissa määrin mukaan. Näin ollen beetaversion tulee olla edustavassa lähes julkaisukelpoisessa kunnossa. Beetaversion tavoitteena on löytää ja korjata viimeisetkin viat projektista ja hioa se julkaisukelpoiseen kuntoon. Beeta koetaan valmiiksi silloin, kun vikoja ei enää löydetä ja markkinointi on valmistellut tuotteen julkaisun.

Julkaisun jälkeen on osa projektiryhmästä vielä valmiudessa reagoimaan mahdollisiin ongelmiin, joita testausvaiheissa on jäänyt huomioimatta. Jos vikoja ei kuulu parin viikon sisään julkaisusta, voidaan projektiryhmä irrottaa kokonaisuudessaan muihin projekteihin. Mahdollisiin vikoihin reagoidaan jatkossa teknisen tuen voimin, joka minun yrityksessäni koostuu hallintaryhmän päivittäisestä foorumien päivystyksestä. Toistaiseksi käytössäni ei ole tekniseen tukeen omistautunutta henkilöä.

4.3 Konkreettiset muutokset

Tuotantoprosessien mallit toimivat itsestään hyvin. Ongelmana oli löytää itselle sopiva ratkaisu. Tarvitsin ketterää menetelmää projektin elävien muutosten takia, mutta silti tarpeeksi dokumentointia, ettei projekti käy epäselväksi.

Ratkaisuksi koin ottaa käyttöön Scrumin erinomaisen vaiheittaisen kehityksen ja vesiputousmallin tyylisen dokumentoinnin, johon lisättävät ominaisuudet otetaan pelialalla käytetystä konseptin mallista (9). Päädyin siis yhdistämään kolmea erilaista tuotantoprosessinmallia päästäkseni yritykselleni sopivaan lopputulokseen.

5 YHTEENVETO

Työn tavoitteena oli kartoittaa aloittavalle pelialan yritykselle sopivat tuotantoprosessin menetelmät. Kartoitusta tehdessä otettiin huomioon yrityksen käytössä olevat resurssit, jotka vaikuttivat huomattavasti lopputulokseen. Ideana oli löytää tasapaino suunnittelun ja toteutuksen välillä samalla selvittäen projektin hallintaan liittyviä seikkoja.

Tietoa etsiessäni huomasin, että pelinkehityksestä on lähes yhtä monta mallia kuin kehittäjiäkin. Varsinaista standardia pelin kehitykseen ei löytynyt. Vaikutti kuitenkin siltä, että valtaosa järjestelmällisistä kehittäjistä noudattivat samantyylistä prosessia.

Lopputuloksena päädyin itsekin noudattamaan samantyylistä prosessia. Sovelsin kaikkea koulussa oppimaani ohjelmistotuotannosta yleisiin pelialan menetelmiin ja lopputuloksena oli selkeä kuva pelin tuotannosta.

Lopputulos olisi voinut kattaa enemmänkin, mutta projektiryhmäni oli selkeästi alimiehitetty. Yritykseni kasvaessa minun kannattaa muuttaa tuotantoprosessia vastaamaan enemmän Scrumin mallia ryhmäkokoonpanon näkökulmasta. Tästä huolimatta olen todistanut useamman yhden miehen peliyrityksen onnistumisen, joten toiveeni ovat korkealla.

LÄHTEET

1. Nousiainen, Eero 2012. Oliosuuntautunut analyysi ja suunnittelu T740203 kurssimateriaali. Oulun seudun ammattikorkeakoulu, tekniikan yksikkö.
2. Ivanof, Antti. Tuotekehitysprosessi ja kahden erilaisen prosessimallin vertailu. Saatavissa: http://www.saunalahti.fi/~ivanoff/tiha/tuotekeh_vert.html. Hakupäivä 13.5.2015.
3. What is Agile model – advantages, disadvantages and when to use it. ISTQB Foundation. Saatavissa: http://istqbexamcertification.com/what-is-agile-model-advantages-disadvantages-and-when-to-use-it/. Hakupäivä 21.5.2015.
4. SCRUM GUIDES 2013. The Scrum Guide. Saatavissa: http://www.scrum-guides.org/scrum-guide.html#theory. Hakupäivä 14.4.2015.
5. Lakeworks 2009. The scrum process. Saatavissa: http://en.wikipedia.org/wiki/Scrum_%28software_development%29#/media/File:Scrum_process.svg. Hakupäivä 14.4.2015.
6. Extra Credits 2015. Making Your First Game: Basics. Saatavissa: https://unity3d.com/learn/tutorials/modules/beginner/your-first-game. Hakupäivä 27.3.2015.
7. Chacon, Scott. – Straub, Ben 2014. Getting started About Version Control. Git. Saatavissa: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control Hakupäivä 14.4.2015.
8. McConnell, Steve 2004. Code Complete, Second Edition.
9. Game Concept. Saatavissa: http://www-rohan.sdsu.edu/~stewart/cs583/game-design/Game_Concept.htm Hakupäivä 2.5.2015.
10. Edwards, Ralph 2006. The game production pipeline: Concept to completion. Saatavissa: http://www.ign.com/articles/2006/03/16/the-game-production-pipeline-concept-to-completion?page=1 Hakupäivä 13.4.2015.

11. Keith, Clinton 2010. Agile Game Development With Scrum: Teams. Gamasutra Saatavissa: http://www.gamasutra.com/view/feature/6040/agile_game_development_with_scrum_.php?print=1 Hakupäivä 14.4.2015.
12. Driessen, Vincent 2010. A successful Git branching model. Saatavissa: http://nvie.com/posts/a-successful-git-branching-model/ Hakupäivä 22.1.2015.

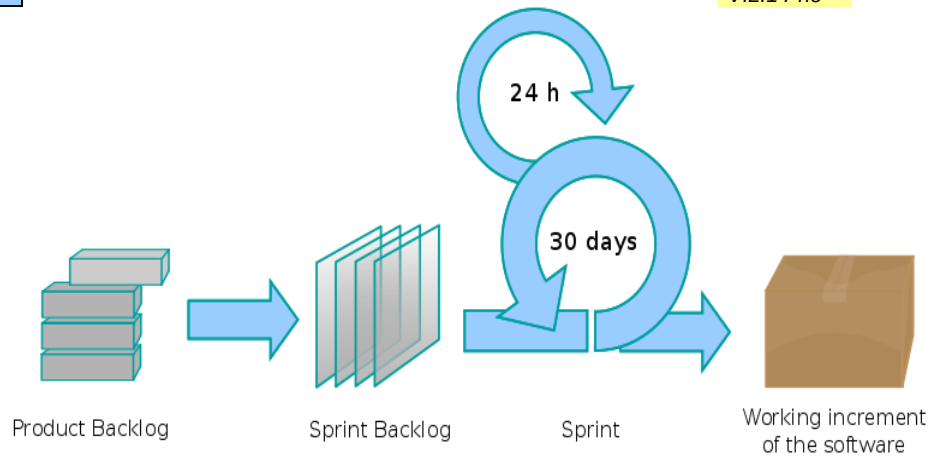
SCRUM-EXCEL-POHJA

Name of the company	Media Nox Software
Team no	team xx
Members of the team	Pekka Kulju
Schedule	22.09.2014 – 18.12.2014
Name of the document	scrum_follow_up.xlsx
Name of the table	Usability of resources

Member	Working days during the project	Other projects	Vacations	Others	Days for the project	Hours for the project	scrum master
Pekka Kulju	64,0	0,0	2,0		62,0 0,0	465,0 0,0	
Totally	**64,0**	**0,0**	**2,0**	**0,0**	**62,0**	**465,0**	

Company:	Media Nox Soft-ware
Project:	On The Move
Project's schedule:	10.03.2014 – 05.05.2014

ID	Product Backlog	Release 1:n jul-kaisu
	very high priority	*V.2.1 Phe*
1	Toimiva testikartta (TM-0003)	
2	Vuoropohjainen toimiminen (TM-0002)	
3	Yksikön hyökkääminen(TM-0004)	
4	Yksikön liikkuminen(TM-0020)	
10	Yksikkörakenne	
	high priority	
5	Testi tekoäly	
6	Taistelun lopettaminen (TM-0007, TM-0015)	
7	Yksikön statistiikka(TM-0001)	
12	Maailman kartta (KM-0001)	



	medium priority
8	Yksiköiden muodostelmat (TM-0009)
11	Yksiköiden varustus(TM-0006)
13	Maailman kartalle generoidaan tehtäviä (KM-0003, KM-0004)
	low priority
9	Yksikkötyyppi (TM-0005)

Company: Oamk
Project: On The Move
Project's schedule: 10.1.2012 - 27.4.2012

Release plan	
weeks in project	26
hours per week	75
total hours in project	888,8
sprints in project	6
release goal	*Pelimekaniikan testiversio pelistä*
sprint 1 schedule	Wk40-41
weeks in sprint 1	2
hours in sprint 1	160
sprint 1 on goal	*Taistelumekaniikan testiversio*
sprint 2 schedule	Wk42-43
weeks in sprint 2	2
hours in sprint 2	160
sprint 2 on goal	*Taistelumekaniikan alfaversio ja maailmankartan testiversio*

sprint 2 schedule	Wk44-45
weeks in sprint 2	2
hours in sprint 2	160
sprint 2 on goal	*Toimiva taistelumekaniikka ja maailmankartan kaupunki näkymä*

sprint 2 schedule	Wk46-47
weeks in sprint 2	2
hours in sprint 2	160
sprint 2 on goal	*Maailmankartan resurssien hallinta*

sprint 2 schedule	Wk48-49
weeks in sprint 2	2
hours in sprint 2	160
sprint 2 on goal	*Maailmankartan kaupankäynti ja quest generaattori*

sprint 2 schedule	Wk50-51
weeks in sprint 2	2
hours in sprint 2	80
sprint 2 on goal	*Mekaanisesti toimiva peli ja QA testaus*

”Toimiva”

Release Backlog

Sprint 1		**Effort as story points**
ID	***Sprint 1 items:***	**1**
1	Toimiva testikartta	1
2	Vuoropohjainen toimiminen	
3	Yksikön hyökkääminen	
4	Yksikön liikkuminen	
5	Testitekoäly	
6	Taistelun lopettaminen	
7	Yksikön statistiikka	
8	Yksikön muodostelma	
9	Yksikkötyyppi	
10	Yksikön rakenne	
Sprint 2		
ID	***Sprint 2 items:***	**12**
5		5
6		3
7		2
8		2

Effort totally

Story Pointseilla voidaan vertailla eforttia sprintin ja eri sprinttien itemien välillä. ***13***

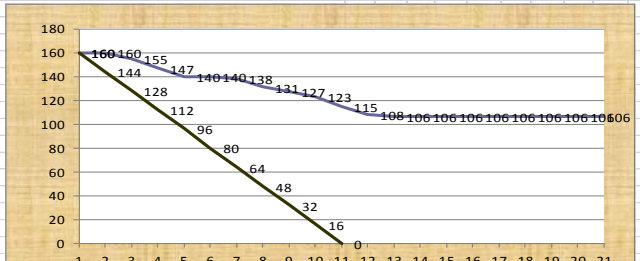
Company: Oamk
Project: On The Move
Project's schedule: 10.1.2012 - 27.4.2012

			w orkload per day in Sprint I (estimated done in 7 days by 2 persons)	8	8	8	8	8	8	8	8		8	8		8	8		8	8		8	8								
			Sprint I Backlog																												
				ma	ti	ke	to	pe	ma	ti	ke	to	pe	ma	ti																
Ids of Sprint I	***Task ID***		Workingdays in Sprint	29	30	1	2	3	6	7	8	9	10	13	14																
		estimated (h) per task	Description of the tasks:	*team hours per working day*																											**Σ**
1			**Toimiva testikartta**																												
	1.1	1	Testipohja		1																										1
	1.2	1	Esteitä		1																										1
	1.3	1	Navigaatio layer		1																										1
2			**Vuoropohjainen toiminen**																												0
	2.1	1	Vuoron vaihto									2	4																		6
	2.2	1	Vuoron listaus		1		1																								2
	2.3	8	Toimintojen toteuttaminen							1			2																		3
	2.4	2	Toimintojen jonottaminen																												0
3			**Yksikön hyökkääminen**																												0
	3.1	8	Hyökkäys RNG																												0
	3.2	3	Hyökkäyksen toteuttaminen																												0
																															0
4			**Yksikön liikkuminen**																												0
	4.1	1	Koordinaatteihin navigoiminen			5								2																	7
	4.2	4	Esteiden väistäminen (muodostelma)											1																	1
	4.3	3	Kiintopisteiden välillä navigoiminen																												0
	4.4	12	Muodotelman preview																												0
5			**Testitekoäly**																												0
	5.1	2	Kohteeseen liikkuminen				2		1																						3
	5.2	16	Kohteeseen hyökkääminen						1	2																					3
	5.3	2	Vuoron vaihto										1																		1
	5.4	16	Kohteen valinta		1								1																		2
6			**Taistelun lopettaminen**																												0
	6.1	4	Tavoite (vastustajan tuhoaminen)																												0
																															0
																															0
7			**Yksikön statistiikka**																												0
	7.1	2	Oletus statistiikka			2	1																								3
	7.2	4	Perusvaikutus (liikkumis, hyökkäys)				2																								2
	7.3	12	Varusteiden vaikutus (simple)																												0
																															0
8			**Yksikön muodostelma**																												0
	8.1	2	"häröpallo"																												0
	8.2	8	Perusmuodostelma (suorakulmio)																												0
9			**Yksikkötyyppi**																												0
	9.1	16	Varustuksen vaikutus																												0
	9.2	8	Yksikön rakenne																												0
	9.3	4	Yksilön luonti																												0
10			**Yksikön rakenne**																												0
	10.1	8	Johtaja(ES)			1	1																								2
	10.2	2	Jäsen									2		2																	4
	10.3	4	Jäsenen paikka											2	2																4
	10.4	4	Line of sight							4	4																				
				0	5	8	7	0	2	7	4	4	8	7	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	54
Task Status																															
			Workload left	160	155	##	140	140	138	131	##	123	115	##	106	##	##	106	106	106	106	106	106								

(workload left = työtä jälellä sprintissä; **sininen käyrä**, piirretty vain kuluvaan päivään asti)

				160	160	155	##	140	140	138	131	##	123	115	##	106	##	##	106	106	106	106	106	106								
Estimated workload left (vihreä käyrä)				160	144	128	##	96	80	64	48	32	16	0																	

Burndown chart for workload of the Sprint 1



LAATUVAATIMUKSET

Laatinut: Kulju Pekka
Pvm: 2.6.2015

Vaatimukset perustuvat lähtökohtaisesti pelikriitikkojen väitöksiin. Vaatimuksissa ei huomioida lisensseihin perustuvia vaatimuksia, vaan kyseessä on asiakkaan mukavuuteen pohjautuvat väitteet.

ID	VAATIMUS	PERUSTELU	LÄHDE
LV-1	Pelin tulee pystyä ylläpitämään vähintään 60 fps:ää tasaisesti PC-alustalle kehittäessä.	Nykyaikaiset PC-alustalla pelaavat ovat tottuneet korkean fps:n mahdollisuuksiin. Korkea fps vaikuttaa pelimukavuuteen muun muassa sulavuudellaan.	Cynical Brit – Gaming critic
LV-2	Pelin tulee pystyä mukauttamaan graafinen ulkonäkö vähintään keskivertojärjestelmän (2–4 vuotta vanhan) suorituskyvylle kelpoiseksi.	Saavuttaakseen mahdollisimman suuren asiakaskannan tulee pelin toimia mahdollisimman monella järjestelmällä.	Yrityksen markkinointisuunnitelma.

LV-3	Ensimmäisestä kuvakulmasta kuvattujen pelien tulee pystyä mukauttamaan field of view enintään 120 astetta.	On todistettu, että jotkut pelaajat kokevat pahoinvointia näkökentän ollessa liian suppea.	Cynical Brit – Gaming critic
LV-4	Värisokeille mukautettu värimahdollisuus.	Viimeaikoina on värisokeat huomioitu useassa pelissä. Vaatimus ei ole pakollinen, mutta sillä saadaan aikaan positiivista huomiota.	Cynical Brit – Gaming critic
LV-5	Painikkeiden uudelleenmäärittäminen.	PC-alustalla on tärkeää, että pelaaja voi mukauttaa käyttöliittymänsä haluamaansa muotoon jo pelkästään painikkeiden määrän takia.	Cynical Brit – Gaming critic
LV-6	Toimintapelejä kehittäessä pelimekaniikka tulee kehittää kontrollien reagointi mielessä.	Responsiiviset kontrollit edistävät pelimukavuutta toimintapeleissä ja aiheuttavat usein	Cynical Brit – Gaming critic

		positiivista palautetta.	